



# Конзолни Артове

Мартин У, Никола, Ясен, Григор, Цветан



# Какво са конзолните икувства?

Конзолата е твърде скучна. Конзолното изкувство е нещо интересно, направено в конзолата, само с принтене на стрингове по подходящ начин. Някои правят дори игри в конзолата. Днес ще изградим система за създаване на конзолни артове, като направим рисуването в конзолата лесно.

# Цветен текст

- Как оцветяваме текст в Java?
- `String text = "Hello World";`
- `//Нормален безцветен текст "Hello world"`
- `String redText = "\u001B[31mHello World";`
- `//Този текст ще е червен "Hello world"`
- Прибавяйки цветни кодове в началото на стринг, ти го оцветяваш. Кода не се вижда когато изпринти.

- `public static final String reset = "\u001B[0m";`
- `//return to default color`
- ```
public static final String black = "\u001B[30m";    //choose color
public static final String red = "\u001B[31m";
public static final String green = "\u001B[32m";
public static final String yellow = "\u001B[33m";
public static final String blue = "\u001B[34m";
public static final String purple = "\u001B[35m";
public static final String cyan = "\u001B[36m";
public static final String white = "\u001B[37m";
```
- `System.out.println(green+"Random Text");` //Текста ще е зелен

# Мразя да пиша "System.out.print"

- Да. Много е досадно. Можем да го съкратим, защото ще го ползваме много сега.

```
• public static void p(String s){           //easy print
    System.out.print(s);
}

public static void l(String s){           //easy println
    System.out.println(s);
}
```

```
public static final String red = "\u001B[31m";
//...
public static final String blue = "\u001B[34m";
//...
public static final String green = "\u001B[32m";
public static final String yellow = "\u001B[33m";
//...
public static final String blue = "\u001B[34m";
public static final String purple = "\u001B[35m";
public static final String cyan = "\u001B[36m";
//...
public static final String white = "\u001B[37m";
```

```
for(int y = 0; y < 10; y++){
    for(int x = 0; x < 10; x++){
        if(x+y > 10){
            p(red+"#");
        }else{
            p(blue+"#");
        }
    }
    l("");
}
```

```
#####
#####
#####
#####
#####
#####
#####
#####
#####
#####
```

Ок а сега 2D поле

# Ъъъ...? Какво се случи току що?

- За тези които не разбраха:
- Едно 2D поле има X и Y координати.
- Да кажем че полето е 10X10 или 10 в X и 10 в Y.
- Обикаляме всяко поле и за него проверяваме дали неговите координати общо правят повече от 10 и после просто оцветяваме.
- Така разделяме полето на 2 - червено и синьо.

# Ама това беше проста вертикална линия.

## Аре нещо по сложно

- Ще направим синусоида. Да. Всъчко което трябва да направим е да увеличим полето и да променим формулата.
- Вече няма да е  $X+Y > 10$  , а ще е  $\text{Math.sin}(x) > y$ .
- ```
for(int y = 0; y < 10; y++){
```
- ```
  for(int x = 0; x < 10; x++){
```
- ```
    if(Math.sin(x)>y){
```
- ```
      p(red+"#");
```
- ```
    }else{
```
- ```
      p(blue+"#");
```
- ```
    }
```
- ```
  }
```
- ```
  l("");
```
- ```
}
```
- ООООО! Виждате ли че от горе са се оцветили през 3 в червено!!!!

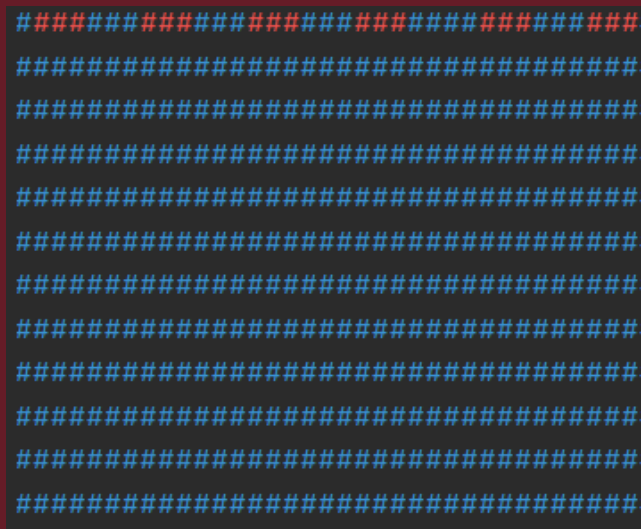
```
#####  
#####  
#####  
#####  
#####  
#####  
#####  
#####  
#####  
#####
```



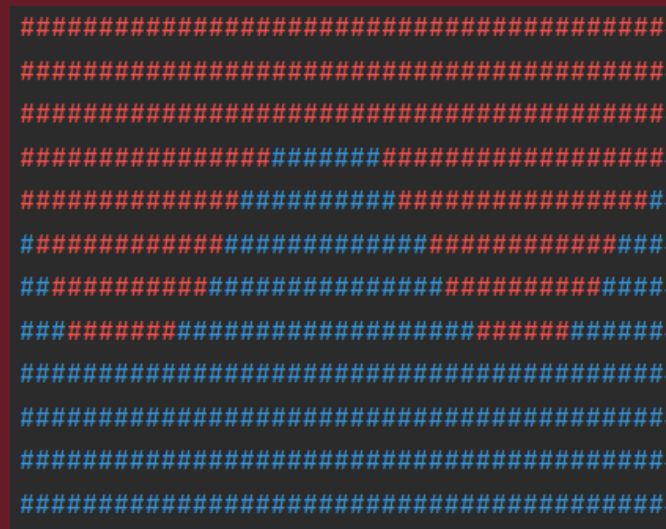
# А... и вие сте разочаровани...

- Добре, да това не прилича на синусида.
- Ама реално е. Просто е много мъничка.
- Сега ще я оголемим.  $x/4$  за да овеличим периода,  $*3$  за височината,  $+5$  за да стои в центъра.

`Math.sin(x)>y`



`Math.sin(x/4)*3+5>y`



# Кадри

За да симулираме дрижение, ние трябва да създадем система за кадри. Кадрите са прерисуване на картината на определени интервали от време. Ако картината бъде прерисъвана с нови параметри, тя ще се промени. Ще пробваме да накараме синусоидата да се движи като вълни. Единственото което ни трябва е java код за изпълняване на дадено парче код на даден интервал време.

# Интернет/Chat GPT има отговор.

Една от опциите е да ползваме класа Timer и TimerTask.

```
public static void main(String[] args){  
  
    Timer timer = new Timer();  
    TimerTask task = new TimerTask() {  
  
        //Само променливи записани тук или променливи които не променят стойността си могат да бъдат ползвани в run() по долу  
  
        int n = 0;  
  
        @Override  
        public void run() {  
  
            //кода тук ще бъде изпълняван на всеки 1000 милисекунди (1 секунда)  
  
            l(n) //принтим n на нов ред веднъж на всяка секунда  
  
            n++ //всяка секунда овеличаваме n с едно  
  
        };  
  
    };  
    timer.scheduleAtFixedRate(task, 0, 1000); //тук може се смени на колко милисекунди да се изпълнява run()  
};
```

Това е начина по който се ползва класа. Това е порграма, която отброява секъндите.

# Да зглобим всичко до тук

- ```
import java.util.Timer;
import java.util.TimerTask;
public class colorfullConsole {
    public static final String reset = "\u001B[0m";    //return to default color
    public static final String black = "\u001B[30m";    //choose color
    public static final String red = "\u001B[31m";
    public static final String green = "\u001B[32m";
    public static final String yellow = "\u001B[33m";
    public static final String blue = "\u001B[34m";
    public static final String purple = "\u001B[35m";
    public static final String cyan = "\u001B[36m";
    public static final String white = "\u001B[37m";

    public static void p(String s){    //easy print
        System.out.print(s);
    }

    public static void l(String s){    //easy println
        System.out.println(s);
    }
}
```

- `public static void main(String[] args){`
- `Timer timer = new Timer();`  
`TimerTask task = new TimerTask() {`
- `//Само променливи записани тук или променливи които не променят стойността си могат да бъдат ползвани в run() по долу`
- `int n = 0;`
- `@Override`  
`public void run() {`
- `//кода тук ще бъде изпълняван на всеки 1000 милисекунди (1 секунда)`
- `for(int y = 0; y < 10; y++){`
- `for(int x = 0; x < 10; x++){`
- `if(Math.sin(x/4+n)*3+5>y){`
- `p(red+"#");`
- `}else{`
- `p(blue+"#");`
- `}`
- `}`
- `l("");`
- `}`
- `n++ //всяка секунда овеличаваме n с едно`
- `};`
- `};`  
`timer.scheduleAtFixedRate(task, 0, 1000); //тук може се смени на колко милисекунди да се изпълнява run()`
- `};`
- `}`
- `}`



# Това бяха снимки на различните кадри.

- Идеята е всеки кадър да измества старите и да се вижда само той в конзолата.
- Обаче още нямаме пълна свобода. Искме да можем да оцветяваме точно където искаме без никакви сложни формули. Искаме дори да можем да видим какъв цвят е определена клетка. За това вече ще съхраняваме терена като двумерен масив от стрингове. После ще ни трябват и методи за лесно оцветяване на голямо количество клетки.

# Полето като матрица

- `public static String[][] arr = new String[160][12];` //160 - размер по x, 12 - размер по y
- `public static void fill(int startX, int startY, int endX, int endY, String color){`  
    //метод за запълване на определена част от терена с определен цвят  
    `for(int x = startX; x < endX; x++){`  
        `for(int y = startY; y < endY; y++){`  
            `if(x<arr.length && y<arr[1].length&&x>=0&&y>=0){` //проверка дали има такава позиция в масива  
                `arr[x][y] = color;`  
            `}`  
        `}`  
    `}`  
}
- `fill(0,0,arr.length,arr[1].length,white);` //до сега матрицата беше празна. Вече е пълна със сомо бяло



- `int maxX = arr.length; //размерите на терена`  
`int maxY = arr[0].length;`
- `for (int y = 0; y < maxY; y++) {`
- `//това ще се случва в повтарящар се run() функция`  
`for (int x = 0; x < maxX; x++) {`  
`String element = "";`  
`element+= arr[x][y];`  
`element += "@"; //string (one element recommended)`  
`element += reset;`  
`p(element);`  
`}`  
`l("");`  
`}`



# С това, което имаме сега можем просто да напишем:

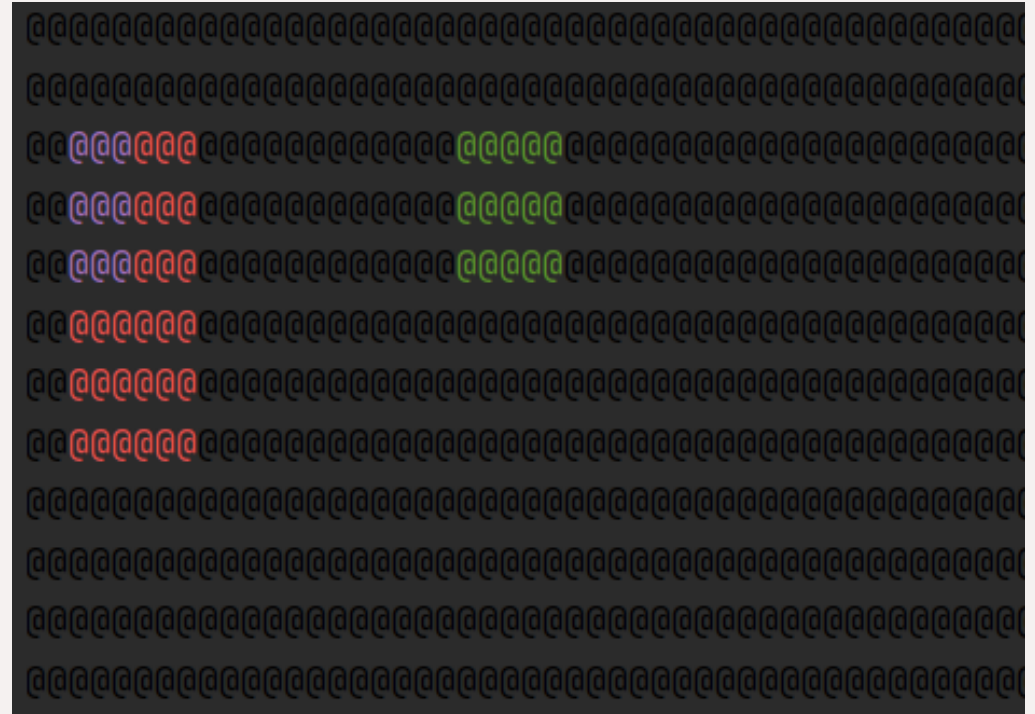
```
fill(0,0,maxX,maxY,black);
```

```
fill(20,2,25,5,green);
```

```
fill(2,2,8,8,red);
```

```
fill(2,2,5,5,purple);
```

//първите два параметъра са координатита на началната точка, вторите два са координатите на крайната точка и накрая е цвета



# fill Better

- Ето един по добър fill метод - по тежък, но по лесен за ползване. Служи за раззширение на нормалния fill:

```
• public static void fillB(double startX, double startY, double endX, double endY, String color){  
    double oldX1 = startX*2.5;  
    int x1 = (int) oldX1;  
    double oldX2 = endX*2.5;  
    int x2 = (int) oldX2;  
    int y1 = (int) startY;  
    int y2 = (int) endY;  
    x2 += x1;  
    y2 += y1;  
    if(x1 < x2){  
        if(y1 < y2){  
            fill(x1,y1,x2,y2,color);  
        }else{  
            fill(x1,y2,x2,y1,color);  
        }  
    }else{  
        if(y1 < y2){  
            fill(x2,y1,x1,y2,color);  
        }else{  
            fill(x2,y2,x1,y1,color);  
        }  
    }  
}
```

- Разликата е в параметрите - началната точка и после колко след нея. Също работи в double. Също прави размерите са  $1x = 1y$ . Преди беше  $2.5x = 1y$ .

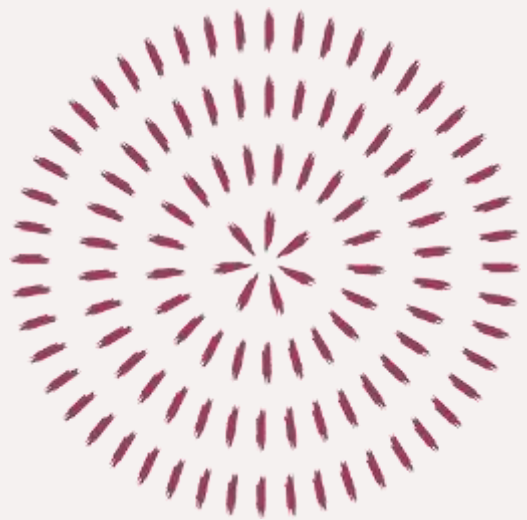
```
• fill(20,2,25,5,green); == fillB(20,2,5,5, green);
```

**ОК. Минахме през всички нови елементи! Ето целия сглобен код, директно готов за копиране:**

- <https://github.com/MartinUzunov/JavaColorConsole/blob/main/colorful.java>

Последвайте ме в Github pls 

Ще е добре всеки да продължи с този код. Сега ще правим по луди неща! След като вече можем да "държим молив", ще се учим да рисуваме с него.



# Векторы и Тригонометрия



# В новия код ще видите:

```
//INIT----->
```

...

```
//----->
```

```
//UPDATE & DRAW----->
```

...

```
//----->
```

- С две думи вече не пипаме другата част от кода. Представете си че това са изолирани контейнери и всеки има собствена роля.
- Е, добре може да пипнете и някоя друга част от кода, но предимно само за да смените някой параметър, като размера на полето или например да добавите друг цвят или нещо такова, но иначе едва ли ще ви трябва да пипате другата част от кода.
- Да се върнем към двата контейнера.
- INIT - в този контейнер декларирате променливи.
- UPDATE & DRAW(или Фрейм за по кратко) - Тук пишете кода, който искате да се повтаря на всеки фрейм.
- Имайте предвид че в края на всеки фрейм, цвета на всичко се връща до //background color, който се намира над UPDATE & DRAW.
- Препоръка: Работете в double и ползвайте fillB, maxXB, maxYB.

# //INIT----->

- double n = 0;

//Май няма да променяме init повече



# Движеща се точка

- `//UPDATE & DRAW----->`
- `fillB(n, maxY/2, 1,1,red);`
- `n++;`

# Кръг

//UPDATE & DRAW----->

```
for(double a = 0; a < Math.PI*2; a+=Math.PI/100){  
    fillB(maxXB/2+Math.cos(a)*5,maxYB/2+Math.sin(a)*5,1,1,blue);  
}
```

Math.PI\*2 - пълна окръжност в радиани

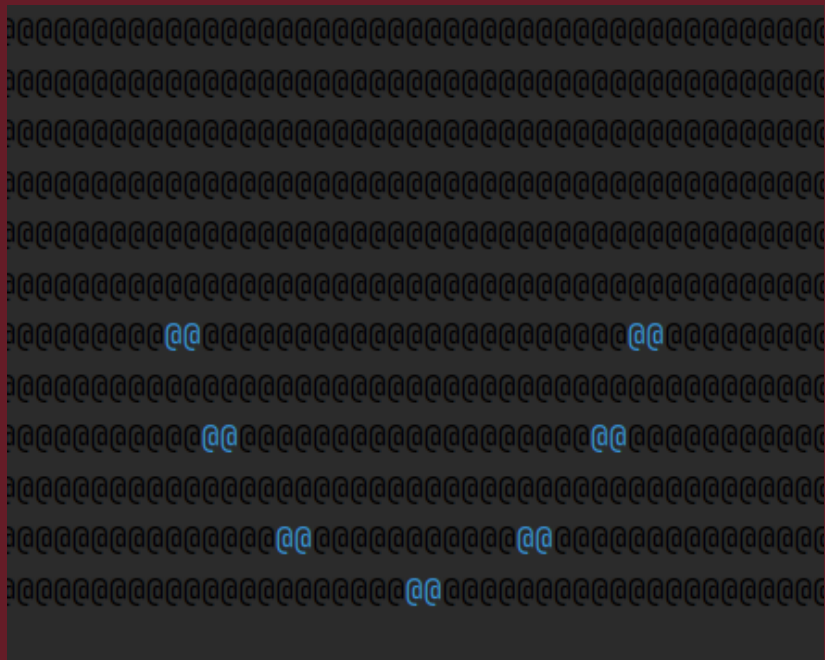
Math.PI/100 - колко плътна да е линията

Math.cos/sin(a) - спрямо ъгъла "a", къде ще се намира новата точка

\*5 - радиуса на окръжността

# С две думи за всеки ъгъл рисуваме по една точка

- ```
for(double a = 0; a < Math.PI*1; a+=Math.PI/6){  
    fillB(maxXB/2+Math.cos(a)*5,maxYB/2+Math.sin(a)*5,1,1,blue);  
}
```

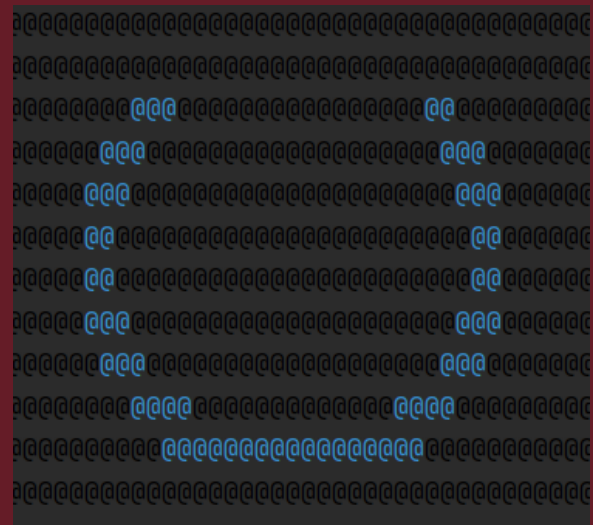
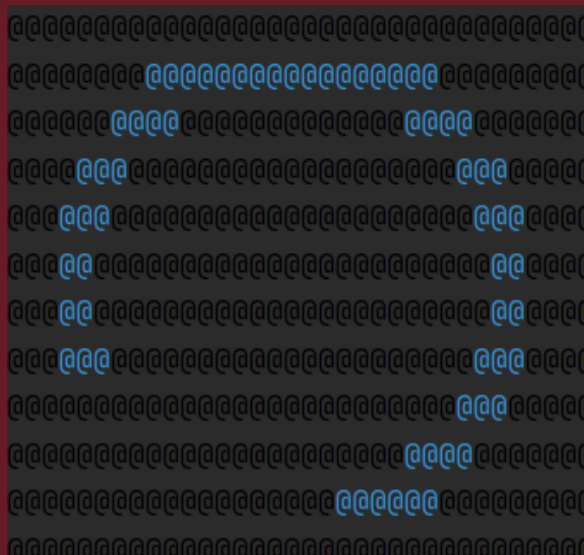
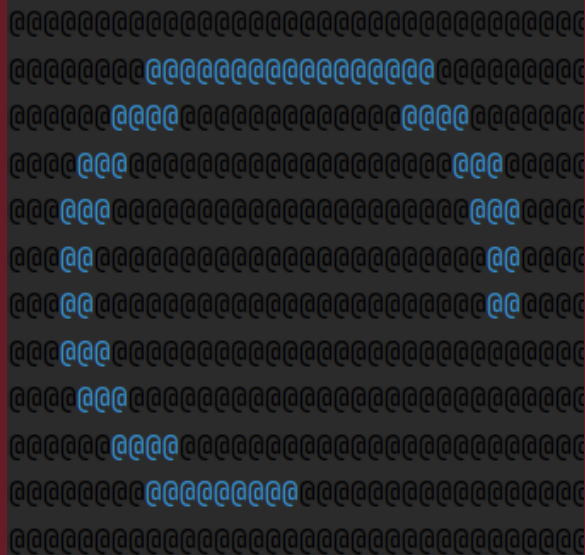


Ако създадем повече ъгли, ще имаме повече точки и съответно по гъста линия. По малко ъгли - по рядка линия. Можем също да не правим пълна окръжност. Може да е полуvin -  $\text{Math.PI} \times 1$ .  $\text{Math.PI} \times 2$  е цяла окръжност.

# Колелцето за зареждане:

```
• for(double a = 0; a < Math.PI*1.5; a+=Math.PI/100){  
    fillB(maxXB/2+Math.cos(a+n)*5,maxYB/2+Math.sin(a+n)*5,1,1,blue);  
}  
n+=0.3;
```

Тук всеки фрейм ъгъла  
е различен заради n

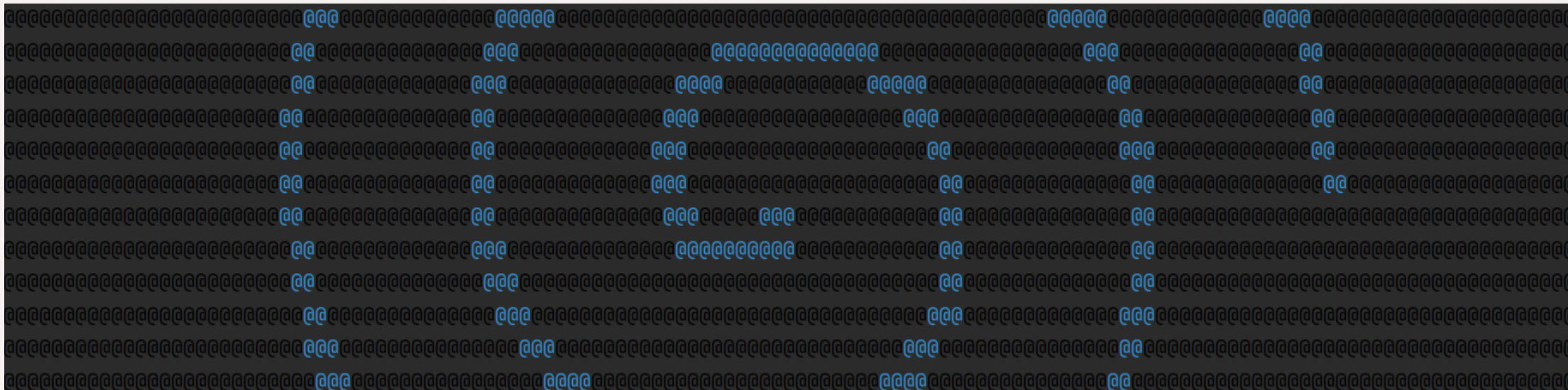




# Спирала

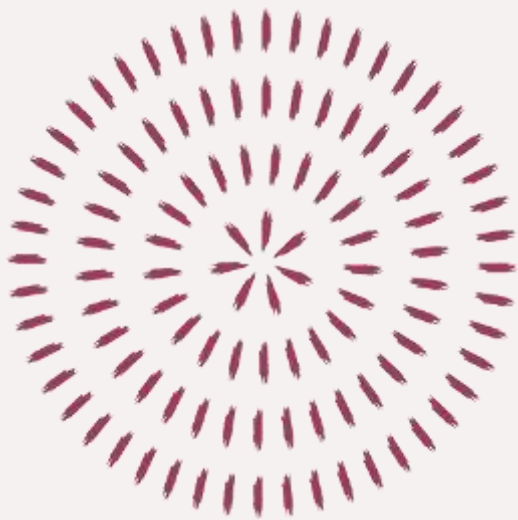
Ако просто радиоса стане "a", ще се получи спирала. Ще овеличим и дължината на 3 окръжности

```
for(double a = 0; a < Math.PI*6; a+=Math.PI/100){  
    fillB(maxXB/2+Math.cos(a)*a,maxYB/2+Math.sin(a)*a,1,1,blue);  
}
```










# Благодаря за вниманието!

Чувствайте се  
свободни да творите  
и други, дори по-  
луди неща, с този  
код!

Последвайте ме в  
Github pls   
<https://github.com/MartinUzunov>

